Sensors and Materials, Vol. 33, No. 2 (2021) 805–814
MYU Tokyo

S & M 2496

Using Microservice Architecture as a Load Prediction Strategy for Management System of University Public Service

Liming Huang,[1] Man-Ying Lee,[2*] Xiaojie Chen,[1]
Hsien-Wei Tseng,[1] Cheng-Fu Yang,[3,4**] and Shun-Fa Lee[2]

[1]College of Mathematics and Information Engineering, Longyan University, Fujian 364012, China
[2]Department of Industrial Economics, Tamkang University, Taipei Tamsui 251, Taiwan
[3]Department of Chemical and Materials Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan
[4]Department of Aeronautical Engineering, Chaoyang University of Technology, Taichung 413, Taiwan

(Received July 24, 2020; accepted January 4, 2021)

Keywords: load prediction, spring boot, microservice architecture, neural network, long short-term memory (LSTM)

The microservice architecture is widely adopted in cloud computing and the applications of software as a service (SaaS), and it can solve problems that the traditional monolithic application development cannot handle. However, new problems, for example, an unbalanced load, appear as many microservice modules coexist in one platform. These modules have complex relationships with each other, causing unavoidable performance bottlenecks. As reported in this paper, we proposed and investigated a load prediction strategy based on long short-term memory (LSTM), which is a revised neural network method, to solve these problems. We used the management system of a university's public service as basic data in the microservice architecture and the Spring Cloud package as the experimental platform. The predicted load trend was compared with the actual load trend to prove that the proposed method can act as a reliable forecasting model. We compared the prediction results of our proposed strategy with those of other classical algorithms, including the autoregressive integrated moving average model (ARIMA), support vector regression (SVR), and LSTM, and we showed that our prediction strategy had higher efficiency than the other methods.

1. Introduction

With the rapid development of cloud computing and Internet of Things (IoT) technologies, more hardware and software resources are being provided by Internet service providers as software as a service (SaaS) schemes. IoT devices represent the sensor layer in an IoT infrastructure, which is responsible for obtaining different types of static/dynamic information about the real world through sensors with different functions. IoT devices can be embedded with software, electronics, and sensors, and it can also be used for feature connectivity with constrained resources. The IoT is a paradigm that allows many smart devices to be connected to the Internet in the cloud, and these devices can be sensors, which can operate and transmit data

*Corresponding author: e-mail: edward@kasus.com.tw
**Corresponding author: e-mail: cfyang@nuk.edu.tw
https://doi.org/10.18494/SAM.2021.3048

ISSN 0914-4935
https://myukk.org/

from a system to other systems. These IoT devices require high extensibility and agility and low fault tolerance in the performance of software systems. Thus, it is important to investigate or develop a system having a backend service architecture that can also have a monolithic architecture in the early stage, for example, a service-oriented architecture (SoA)[1,2] and a microservice-based architecture.[3,4] A microservice-based architecture has a system of real-time environmental sensors, and it is highly scalable for applications in a cloud environment.

The microservice architecture is connected to computational workflows for data analysis because this architecture provides complete, ready-to-run, reproducible data analysis solutions that can be easily deployed in private and public clouds as well as on desktop computers. Moreover, using the bounded context method of the microservice decomposition architecture, design ideas and implementation schemes for resource monitoring and scheduling and for obtaining data synchronization in the cloud have been proposed to provide combat capabilities with high efficiency and fast collaboration.[5] A microservice-based architecture also has many advantages in large-scale software development, which can be divided into many solo applications as small independent services according to business logic.[6,7] Even though the microservice architecture is very popular, it also introduces problems in service communications and resource allocation efficiency when the number of users rapidly increases or large-scale heterogeneous resources are needed. Because there are many microservice modules existing in one platform, which have complex relationships with each other, they can cause fatal performance bottlenecks in the resource allocation of the system, making load prediction an urgent necessity.[8]

There have been many studies focused on the load prediction of the human flowrate of public services, including algorithms based on traditional probability statistics, data mining, neural networks, and deep learning. Load prediction based on traditional probability statistics includes the exponential smoothing model, autoregressive model, and autoregressive moving average model. Monfared *et al.* proposed a new adaptive exponential smoothing method, which can automatically adjust parameters according to reality and the model can be updated immediately over time.[9] Calheiros *et al.* provided an autoregressive moving average model, which is applied in SaaS for cloud computing to predict the physical machine load.[10] However, these methods are all based on predefined formulas, which limit their generalization ability to deal with ongoing load data, making it difficult to accurately depict actual situations with complex conditions. Shahin proposed an adjustable algorithm based on a dynamic threshold, which was constructed with the use of long short-term memory (LSTM) [a variant of the recurrent neural network (RNN)] to predict the size of resources and automatically allocate virtual machines according to predicted values.[11] Qiu *et al.* also developed a deep learning method but with a different bottom model, which was a combination of a multilayer restricted Boltzmann machine and a deep belief network (DBN).[12] These load models based on the above prediction methods were only concerned with hardware execution efficiency because the central process unit (CPU) and memory were the main factors. However, it is possible to neglect the load of elements including the CPU, memory, hard disk I/O, and network I/O. Moreover, a DBN is not good at dealing with time series data, in contrast to the RNN. A gate recurrent unit (GRU) is a variant of LSTM. However, to our knowledge, no research has focused on the load prediction strategy

for the management system of a university public service by forecasting potential problems. In this study, we proposed a new load model, which considered the efficiencies of the CPU, memory, network I/O, and hard disk I/O. We also modified the LSTM model, which was a GRU model, for conducting the network training and used it in our load prediction strategy. We also simplified the gate compared with that in other studies with the aim of resource optimization. Also, to improve the accuracy, especially the processing accuracy of the predicted time series data, we improved the calculation efficiency. We compared the prediction results of our proposed strategy with other classical algorithms, including the autoregressive integrated moving average model (ARIMA), support vector regression (SVR), and LSTM. We found that our prediction method had higher efficiency than the other methods.

2. University Utility Architecture

A university utility is a cloud computing platform based on the microservice architecture, which is a renewed architecture and can be investigated as a common monolithic application. We adopted this scheme because the microservice architecture is relatively suitable for situations where the number of users is rapidly increasing and system maintenance becomes more burdensome without sufficient IT expertise. The services of management systems often need to run for the whole day without interruptions. When services are developed and deployed by different teams using different technologies, it is preferable to choose the microservice architecture for better development & operations (DevOps) practice.[13]

The utility architecture of our university (Longyan University) is shown in Fig. 1. Users can visit the university utility from a cell phone or PC, and the log-in information is intercepted by a common public user interface that is responsible for unified authentication, security, and verification. This utility provides many services in the front end, including score query, school news, financial information, library access, and curriculum information. Users can tap any module that they want, and these requests are sent to the back-end destination. The Zuul

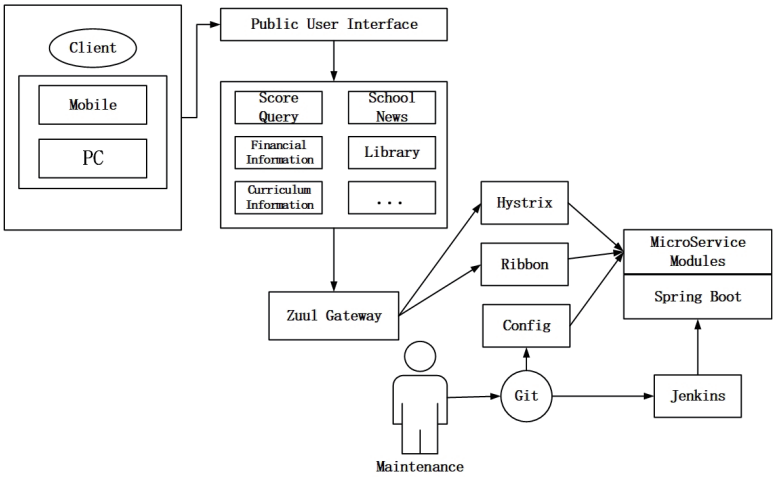

Fig. 1. University utility architecture.

gateway, which is designed for service information routing and location, is a necessary part of the system as a large number of microservice modules exist, resulting in frequent changes in the service IP address. When there are disturbances in microservice modules, which are not uncommon in a large-scale system, Hystrix can act as a fuse and quickly diagnose the locations of disturbances, cut off the related paths, and redirect the requests to alternative service modules. Ribbon is a load balancer, which is a critical component in the architecture. Its main function is to distribute client requests to different targets for microservice modules using its internal self-learning algorithm. With the help of Ribbon, the system load burden can be distributed more equally to each service module by considering its physical host burden. Users can upload related developed service module information into Config using the Git tool, and Config saves all the configuration information and downloads it to each module that requires it whenever necessary. The microservice modules discussed in this paper were developed by Spring Boot, which is a fast and convenient integrated development package, and Jenkins was responsible for microservice module testing and integration testing.[14,15]

3. Improved Load Prediction Strategy

3.1 Load model

So far, load models having functional limitations have often been considered, which are only concerned with the functions of CPU and memory. Actually, the hardware for the load models is more complicated in practice, so the load model in this study considers the parameters of CPU, memory, disk I/O, and network I/O. The load computing formula is described below, in which parameters with the prefix W refer to coefficients and they dominate the use of resources in the load model. $Load_{Cpu}$, $Load_{Mem}$, $Load_{Disk}$, and $Load_{Net}$ respectively refer to the load values of the CPU, memory, disk, and network in one physical host at a given moment after normalization procedures. $Load_{Host}$ is the predicted load of a physical host at a given moment and is calculated as follows.

$$\begin{aligned} Load^{t}_{Host} = {} & W_{Cpu} \cdot Load_{Cpu}{}^{t} + W_{Mem} \cdot Load_{Men}{}^{t} \\ & + W_{Disk} \cdot Load_{Disk}{}^{t} + W_{Net} \cdot Load_{Net}{}^{t} \end{aligned} \tag{1}$$

3.2 LSTM

Neural networks originated in the 1950s, and they have gradually been improved with the development of the perceptron network structure, which is a model consisting of three layers: an input layer, an output layer, and a hidden layer. Data can be incorporated in the input layer and output from the output layer after transformation in the hidden layer. With the improvement of models, many kinds of neural networks with multiple hidden layers have been invented, in which different optimization functions have been used. However, these neural network models are not good at dealing with time series data, and the RNN was developed to overcome this problem.[16] The output of neurons in the RNN can be applied as input data for the next time

stamp, in which the connections of the structure established by the neurons of the RNN can remember input data strings of any length. The RNN can be seen as a kind of deep neural network (DNN) with the depth of its structure equivalent to the time length, which means that the condition of fading away is inevitable. To solve this problem, LSTM, an upgrade of the RNN, was designed, an example of which is the GRU.[17] The critical part of the RNN is the transfer function, which can be described by Eq. (2), where X_t, the input of time stamp t, and $Hidden_{t-1}$, the output of the hidden layer of the prior time stamp $t-1$, can be combined to update the output $Hidden_t$ of the hidden layer at time stamp t.

$$Hidden_t = F\left(X_t,\ Hidden_{t-1}\right) \tag{2}$$

To characterize the properties of the RNN, F is a nonlinear differentiable transfer function and different values of F correspond to different RNN models. In this paper, we choose the vanilla RNN model, which is a simple transformation type of RNN and can be seen below. In Eq. (3), $T_1 \in R^{M\times N}$ and $T_2 \in R^{M\times M}$ are transformation matrixes, $b \in R^M$ is a partial vector, and F is a nonlinear activation function that is suitable for the nonlinear character of the data with common activation functions such as sigmoid and tanh.

$$Hidden_t = F\left(T_1 X_t + T_2,\ Hidden_t + b\right) \tag{3}$$

Because the vanishing gradient problem is a classical problem, i.e., RNN models cannot capture data with long-term reliability, the transfer function will forget early training information when the time series data is long. LSTM is chosen for this situation because it can introduce the gate function in the design of the transfer function. The GRU is a minor revision of LSTM with simplified gates, and in the GRU model, two gates are introduced in neurons. The first one is *reset_ gate*, used for adjusting the combination of the current input data and the prior output hidden memory, and the other is *update_gate*, used for controlling and saving the training memory of the prior time stamp. The transfer function in the hidden layer of the GRU can be expressed as Eqs. (4)–(7), with T_1, T_2, and b co-shared among all time stamps during model training, in which the operator $\odot$ is the dot product. The structures of the RNN and the variations of the GRU neurons are shown in Fig. 2.

$$update\_gate_t = F\left(T_1 X_t + T_2 Hidden_{t-1} + b\right) \tag{4}$$

$$reset\_gate_t = F\left(T_1 X_t + T_2 Hidden_{t-1} + b\right) \tag{5}$$

$$Hidden'_t = Sigmoid\left(T_1 X_t + T_2\left(reset\_gate_t \odot Hidden_{t-1}\right)\right) \tag{6}$$

$$Hidden_t = \left(1 - update\_gate_t\right) \odot Hidden_{t-1} + update_{gatet} \odot Hidden'_t\ ) \tag{7}$$

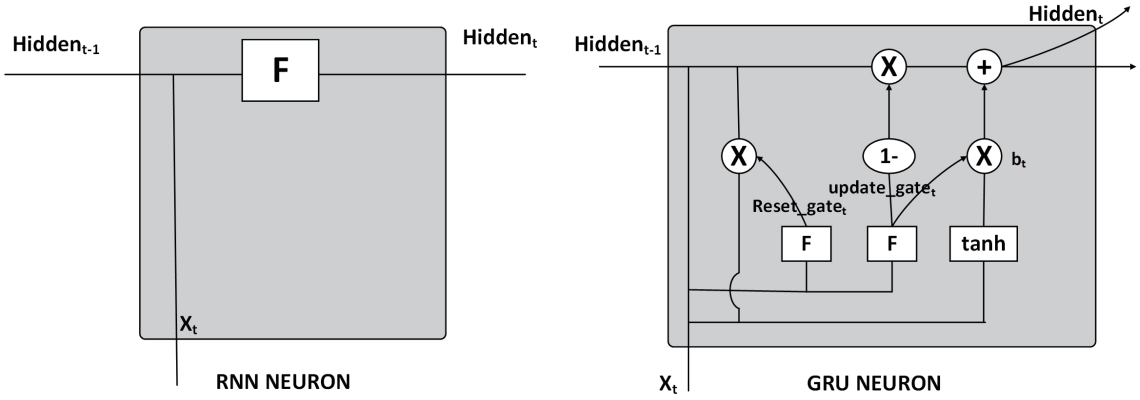

Fig. 2. Structures of RNN and the variations of the GRU neurons.

3.3 Neural network training

Neural network model training focuses on the hidden layer. First, in the input layer, we collect the four kinds of load time-series data mentioned above, which are normalized as follows:

$$\begin{aligned} Load_{Cpu} &= \left(Load_{Cpu}^{1}, Load_{Cpu}^{2}, ..., Load_{Cpu}^{n} \right) \\ Load_{Men} &= \left(Load_{Men}^{1}, Load_{Men}^{2}, ..., Load_{Men}^{n} \right) \\ Load_{Disk} &= \left(Load_{Disk}^{1}, Load_{Disk}^{2}, ..., Load_{Disk}^{n} \right) \\ Load_{Net} &= \left(Load_{Net}^{1}, Load_{Net}^{2}, ..., Load_{Net}^{n} \right). \end{aligned} \tag{8}$$

Then we incorporate the one-dimensional vector data of the CPU, memory, disk, and network into Eq. (1) to form the following time series matrix:

$$Load = \left(Load_1, Load_2, ..., Load_n \right). \tag{9}$$

The training set and testing set in the model training can be expressed as below, with the constraint condition $m < n$.

$$\begin{aligned} Load_{training\_set} &= \left(Load_1, Load_2, ..., Load_m \right) \\ Load_{testing\_set} &= \left(Load_{m+1}, Load_{m+2}, ..., Load_n \right) \end{aligned} \tag{10}$$

Then we incorporate the input data into the classical slide window method, assuming a data length of *length*. The divided model input is expressed as

$$X = \left(X_i, X_{i+1}, ..., X_{length} \right),\ i \in \left[0, length \right]. \tag{11}$$

For this input, the initial prediction output in the output layer can be expressed as

$$Y = \left(Y_i, Y_{i+1}, ..., Y_{length}\right), i \in \left[0, length\right]. \tag{12}$$

Then the initial input data is incorporated into the hidden layer for the training calculation with the Adam optimization algorithm used to continuously update the related parameters in the model until the predefined prediction accuracy is satisfied. The output data processed by the hidden layer can be expressed as Eq. (13), in which S_{t-1} and $Neuron_{t-1}$ are the state and output of the prior GRU neuron, respectively, and GRU refers to the combination of Eqs. (4)–(7).

$$\begin{aligned} Hidden &= \left(Hidden_1, Hidden_2, ..., Hidden_{length}\right) \\ Hidden_t &= \mathrm{GRU}\left(X_t, S_{t-1}, Neuron_{t-1}\right) \end{aligned} \tag{13}$$

As the input data is incorporated into the input layer, the output data will be "hidden" in the hidden layer calculation, and the initial prediction data in the output layer will be an array of length defined as *length*. We adopt the mean square error (MSE) as the error evaluation method, which is a method to calculate the loss, where the loss can be described as follows:

$$Loss = \sqrt{\frac{1}{length\left(m - length\right)} \sum_{t=1}^{length\left(m-length\right)} \left(Hidden_i - Y_i\right)^2} \tag{14}$$

Our load prediction strategy can be generalized as below in a pseudo-format, in which we need to initialize a few parameters, including the load data *load*, error accuracy *o*, possible steps, the GRU neuron state *S*, and the Adam optimization parameter *seed*, before model training. With the help of *prediction data*, we can predict the trend from the recent payload, i.e., whether it will exceed the threshold or is expected to increase in the future, providing useful information for the future scheduling strategy.[18]

3.4 Load prediction strategy

The processes of the load prediction strategy are listed below:

Input Data: <*Load*, *set length*, *o*, *length*, *S*, *seed*, *steps*, *wait for prediction data*$_{length}$>

Output Data: <*prediction data*>

(a) Fetch $\mathbf{Load_{training}}$ and $\mathbf{Load_{testing}}$ data from **Load** by **set-length**
(b) Fetch **X** and **Y** from $\mathbf{F_{training}}$ by **length**
(c) $\mathbf{GRU_{neuron}}$ creation by **S**
(d) $\mathbf{GRU_{network}}$ connection by $\mathbf{GRU_{neuron}}$
(e) $\mathbf{GRU_{network}}$ initialization by **seed**
(f) foreach step in **steps**
(g) **Hidden ← GRU(X)**

(h) **Loss** ← $\sqrt{\frac{1}{length(m-length)}\sum_{t=1}^{length(m-length)}\left(Hidden^{i}-Y^{i}\right)^{2}}$

(i) $\mathbf{GRU_{network}}$ update by **Adam** optimization with **Loss** and **o**

(j) end for

(k) Achieve $\mathbf{GRU_{network\text{-}completion}}$

(l) **Prediction-data** ← $\mathbf{GRU_{network\text{-}completion}}$ (wait for $\mathbf{prediction_{length}}$)

(m) return **Prediction-data**

4. Experimental Results and Discussion

We carried out an experiment to demonstrate the effectiveness of the improved load prediction strategy. In the experiment, we first normalized the initial data from four different sources, the CPU, memory, disk I/O, and network I/O, using Eq. (1). These data were collected in physical hosts by the script program Python. To verify the prediction accuracy of our model, several control experiments were set up with MSE and the mean absolute percentage error (MAPE) chosen as the evaluation indexes, where

$$MAPE=\frac{1}{n}\sum_{i=1}^{n}\frac{\left|Hidden_i-Y_i\right|}{Y_i} \tag{15}$$

Python 3.5, TensorFlow2.0, and Linux CentOS 7.2 were used in our experiment. We chose the length of the load data to be 30, the number of hidden layers in the GRU to be 4, with 50 neurons in each layer, and the number of iterations to be less than 300 to avoid spending too much time on training. The required prediction error was set to 10^{-6}. The trained model, instructed in accordance with the above method, was used for load prediction and compared with the actual load curve, as shown in Fig. 3. Although the curves do not perfectly match, they both show the same trend and have similar data values, which means that the proposed load prediction strategy has good accuracy.

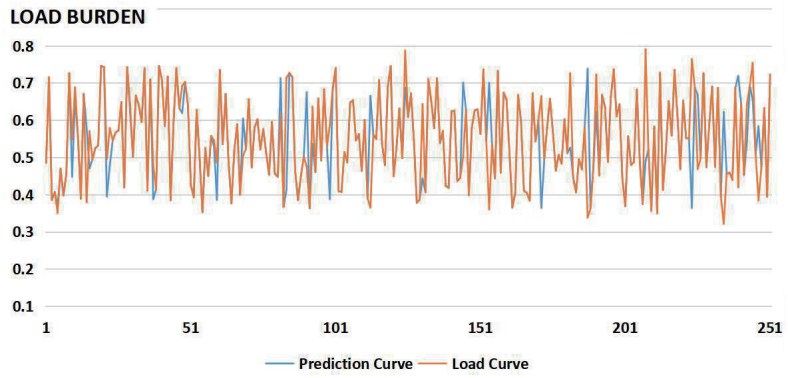

Fig. 3. (Color online) Comparison between predicted load and actual load.

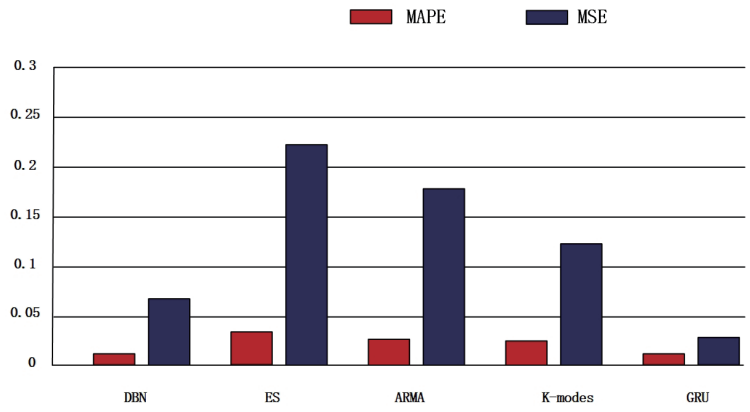

Fig. 4. (Color online) Comparison of results of different strategies.

Then we compared our strategy based on GRUs with some popular prediction strategies for load prediction: DBN, ES, ARIMA, and K-modes. The results are shown in Fig. 4[19,20] and reveal that our GRU model has the smallest prediction loss. This is because ES and ARIMA are based on traditional probability statistics, and it is difficult to fit the complex and changing load data with predefined formulas. DBN has a better performance than ES and ARIMA owing to its strong generalization ability, but it is ineffective for time series data. K-modes requires a large amount of fundamental data for early training but it cannot effectively use preliminary data. This is because the effectiveness of using history data decreases over time, making it difficult to manipulate the data of time series in practice.

5. Conclusions

We proposed an improved load prediction strategy based on the neural network LSTM method and its revised GRU model. We investigated the performance of this method using our university utility platform, which was based on a microservice architecture and Spring Cloud package, and the results showed that it had higher efficiency than the DBN, ES, ARIMA, and K-modes methods. Furthermore, we consider that although the load elements and GRU model achieved the expected improvements, there is still a lot of space for further improvement.

Acknowledgments

This work was supported by the Young and Middle-aged Teacher Program of the Education Department of Fujian (JAT170579 and JAT190741); Longyan University's Qi Mai Science and Technology Innovation Fund Project of Longyan City (2017QM0201 and 2018LYQM0202); Longyan University's Qi Mai Science and Technology Innovation Fund Project of Liancheng County, Longyan; Longyan University's Research and Development Team Fund, Great Project of Production, Teaching, Research of Fujian Provincial Science and Technology Department (2019H6023); and Project Nos. MOST 108-2221-E-390-005 and MOST 109-2221-E-390-023.

References

1 Z. H. Wu, S. G. Deng, and J. Wu: Service Computing: Concept, Method and Technology (Elsevier Inc., 2015). https://woodward.library.ubc.ca/research-help/journal-abbreviations/
2 K. Dirk, B. Karl, and S. Dirk: Enterprise SOA: Service-Oriented Architecture Best Practices (Prentice Hall, Nov. 19, 2004).
3 A. A. Shahi: Int. J. Adv. Computer Sci. Appl. **7** (2016) 279.
4 J. Z. Yan, X. Y. Chen, Y. C. Yu, and X. J. Zhang: Water **11** (2019) 1317.
5 L. Y. Tang, H. R. Hu, Z. H. Wang, J. S. Wang, and Y. H. Li: Int. Conf. Big Data and Security in Big Data and Security (2019) 402.
6 C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis: IEEE Software **34** (2017) 91.
7 F. Rademacher, J. Sorgalla, and S. Sachweh: IEEE Software **35** (2018) 36.
8 G. Aldabbagh, S. T. Bakhsh, N. Akkari, S. Tahir, S. Khan, and J. Cioffi: Wireless Networks **21** (2015) 2413.
9 M. A. S. Monfared, R. Ghandali, and M. Esmaeili: J. Industrial Engineer. Inter. **10** (2014) 209.
10 R. N. Calheiros, E. Masoumi, R. Ranjan, R. Buyya: IEEE Trans. Cloud Comput. **3** (2015) 449.
11 A. A. Shahin: Inter. J. Adv. Computer Sci. Appl. **7** (2017) 279.
12 F. Qiu, B. Zhang, and Jun Guo: 2016 17th IEEE/ACIS Int. Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) (2016) 319.
13 K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber: LSTM: IEEE Trans. Neural Networks Learn. Syst. **28** (2015) 2222.
14 D. L. Giudice, C. Mines, and A. Homan: Forrester Res. (2016) 1.
15 T. Giovanni, B. Sandro, B. Martin, S. Josef, and M. B. Thomas: Future Generation Comput. Syst. **72** (2017) 165.
16 R. Suthat and C. Arpnikanondt: J. Syst. Software **117** (2016) 30.
17 V. Chunwijitra and A. Chotimongko: EURASIP J. Audio Speech Music Process. **16** (2016) 1.
18 Y. Wu, M. Yuan, S. P. Dong, L. Lin, and Y. G. Liu: Neurocomputing **275** (2018) 167.
19 J. Chung, C. Gulcehre, K. H. Cho, and Y. Bengio: NIPS 2014 Deep Learning and Representation Learning Workshop (December 2014).
20 X. Xu, H. Yu, and X. Pei: 2014 IEEE 17th Int. Conf. Computational Science and Engineering (Chengdu, 2014) 257–264. https://doi.org/10.1109/CSE.2014.77